

Using the Redirector and StreamIOH

Paul Kimpel

2007 UNITE Conference
Session MCP-4026

Wednesday, 12 September 2007, 2:45 p.m.

Copyright © 2007, All Rights Reserved

Paradigm Corporation

Using the Redirector and StreamIOH

2007 UNITE Conference
Valley Forge, Pennsylvania

Session MCP-4026

Wednesday, 12 September 2007, 2:45 p.m.

Paul Kimpel

Paradigm Corporation
Poway, California

<http://www.digm.com>

e-mail: paul.kimpel@digm.com

Copyright © 2007, Paradigm Corporation

Reproduction permitted provided this copyright notice is preserved
and appropriate credit is given in derivative materials.

Presentation Topics

- ◆ **Background**
 - File transfer vs. file access
 - File structures
 - Network shares
- ◆ **The Redirector**
 - Using the Redirector
 - Security and user authentication
 - Directory operations
 - Printer Shares and the PCDRIVER utility
- ◆ **StreamIOH**
 - Accessing byte-stream files within the MCP
 - Accessing text files on network shares

Paradigm MCP-4026 2

Today I am going to discuss two facilities in the MCP that I think are really interesting and fun, and that I find I am using more and more all the time – the Redirector and StreamIOH.

- The **Redirector** allows MCP applications to access files directly on shared network directories.
- **StreamIOH** allows MCP applications to read and write line-delimited text files as if they were traditional record-oriented files. StreamIOH can be used by itself for MCP-resident text files, or in conjunction with the Redirector for text files on network shares.

I will begin by discussing some background information – a brief discussion on methods for file transfer vs. those for directly accessing files, an overview of MCP file structures and how they relate to file structures on other systems, and a similarly brief introduction to the concepts behind shared network resources.

The next part of the presentation will focus on the Redirector – how it is implemented, how you use it with MCP applications, how you manage security and authentication with remote servers, how you can access and manipulate remote directories or folders, how to use the Redirector with shared network printers, and a brief overview of a Redirector utility program.

The final part of the presentation will discuss StreamIOH – how it is implemented, how you use it with MCP applications, how it can be used to access text files stored in the MCP file system, and how it can be used to access text files stored on network shares.

I have a few examples and demonstrations to share, and at the end is a list of references where you can learn more about both of these facilities.



Some Background

To begin, I want to cover some background topics that are important in understanding how the Redirector and StreamIOH fit into the MCP environment and can work with typical MCP applications.

File Transfer vs. File Access

- ◆ MCP has several **file transfer** mechanisms
 - BNA File Transfer
 - BNA NFT
 - FTP
 - OSI-FTAM
- ◆ Redirector is a **file access** mechanism
 - File is not just copied to/from the MCP system
 - MCP applications directly read/write the file on the network share

Paradigm MCP-4026 4

The first issue is to distinguish between *file transfer mechanisms* and *file access mechanisms*.

The MCP supports a number of file transfer mechanisms. The first of these were implemented as part of BNA. Along with TCP/IP, we also got FTP and, indirectly, OSI-FTAM. The purpose of these file transfer mechanisms is to move a file from one system to another system over a network, possibly doing some light conversion along the way, such as character translation or record-level formatting.

By way of contrast, a file access mechanism is one that does not transfer a file from one system to another. It allows an application on the local system to access the file directly on the remote system. Reads and writes on the client system take place as soon as possible on the remote server, allowing for any network delay and caching that may be taking place. You can use a file access mechanism to implement a file transfer system, but that is just one example (and a fairly degenerate one) of how file access can be applied.

The Redirector, which we will talk about in more detail, is a file access mechanism. It allows MCP application programs to read and write files directly on remote servers. The file never need be present on the MCP system.

File Structures

- ◆ MCP supports three file structures
 - ALIGNED180 (default, most common)
 - BLOCKED
 - STREAM

- ◆ MCP applications traditionally use record-oriented I/O
 - File system understands internal file record/block structure
 - Programs read/write whole records
 - Rich mechanism for specifying logical and physical file characteristics through attributes

Paradigm MCP-4026 5

Before we can talk about how MCP applications can access files on remote systems, we need to talk about some concepts of the MCP file system. One of the primary concepts is that of *file structure*, as implemented by the **FILESTRUCTURE** attribute. This attribute currently has three mnemonic values:

- **ALIGNED180** – This is the default, and by far most common value. It implies that disk files will be structured using fixed-length blocks, and that the blocks will always be aligned on 180-sector boundaries. Whether the disk has physical 180-byte sectors (such disks have not been manufactured for years now), or one of the virtual sectoring technologies used with modern disks (e.g., VSS1, VSS2, or the VMMCP's Logical Disk files), block alignment is always done on these traditional sector boundaries. The size of the block is also the size of physical I/O transfers to and from the disk. Records within the blocks of **ALIGNED180** files may be fixed or variable length, but a record is always wholly contained in a block. Files of this structure may also be "reblocked" – opened with a different block size than used when the file was created, provided the new blocksize is a multiple of the disk sector size.
- **BLOCKED** – This is similar to **ALIGNED180**, in that a file is composed of fixed-length blocks aligned on physical disk sectors, but the sector size is not constrained to 180 bytes. Files with this structure may not be "reblocked."
- **STREAM** – Files with this structure are not composed of blocks. Records are simply written head-to-tail without any wasted space. Records may span physical disk I/Os, and even may span physical disk area allocations. There is a special case of stream files, termed "byte stream" files, which I will discuss in a minute.

Although the **BLOCKED** and **STREAM** file structures have been available since Mark 3.9 (ca. 1990), most MCP applications today still use the traditional **ALIGNED180** structure. This also implies that these applications use record-oriented I/O. The file system stores information about record and block sizes in the disk directory, and the MCP Logical I/O module uses that information to isolate programs from the physical aspects of blocking, unblocking, and locating individual records in allocated disk areas. Programs simply read and write whole records. The MCP provides a rich mechanism for configuring record-oriented files and interrogating their attributes through the use of file attributes. These attributes are essentially properties of a "logical file" object.

File Structures, continued

- ◆ Other systems use a much simpler model
 - Files are simply "streams" of bytes
 - File system knows nothing about internal file structure
 - Basically two kinds – binary data and text
- ◆ Text (delimited) stream files
 - Traditionally encoded in ASCII or ISO 8859
 - Partitioned into "lines" by delimiter characters
 - DOS/Windows: CR-LF pair
 - Unix/Linux/Mac: LF (called NL or "newline")
 - CTOS/BTOS: LF
 - Mac (pre-OS/X): CR

Paradigm

MCP-4026 6

Most other types of systems, especially non-mainframe systems, use a much simpler file model. Files are structured simply as linear sequences ("streams") of bytes. There are no record or block structures, and the file system knows nothing about the internal structure of files. Internal structure is something that is left entirely up to the application accessing the file. There are basically two general types of files – binary data and text – but the distinction is usually left up to the application, not the file system.

Text files are a special, but very common, case of stream files. These files are typically encoded in ASCII or one of the nationalized variants of the ISO 8859 standard. The file is logically partitioned into variable-length "lines." Lines are delimited from each other by certain characters. The delimiter characters vary by operating system, with the most common choices being line-feed (ASCII hex **0A**, often called "new-line" or NL) for Unix/Linux systems, and the carriage-return/line-feed pair (ASCII hex **0D0A**) for Microsoft Windows/DOS systems. Carriage-return alone is used by some systems, particularly pre-OS/X Apple Macintosh systems.

MCP Stream Files

- ◆ MCP Logical I/O supports stream files
 - Record streams
 - Byte streams
- ◆ Byte-stream files correspond to the unstructured stream files of other systems
- ◆ Therefore, MCP apps must access files on network shares as byte streams

Paradigm

MCP-4026 7

As mentioned earlier, the MCP supports the **STREAM** file structure. There are two forms of stream files, record streams and byte streams, with byte streams being a special case of record streams. These byte-stream files correspond to the unstructured files on other systems, particularly Windows and Unix/Linux systems.

Therefore, when MCP applications access files directly on remote systems, they must do so as byte-stream files and program for them accordingly. From this point on in the presentation, when I mention "streams" or "stream files," I will mean byte streams, as record streams are an MCP-specific concept.

There is a way around this requirement that MCP applications access remote files as streams, as we will see when we get to the discussion on StreamIOH.

Declaring MCP Byte Stream Files

- ◆ Required attributes
 - **FILESTRUCTURE** = **STREAM**
 - **FILEORGANIZATION** = **NOTRESTRICTED** (default)
 - **BLOCKSTRUCTURE** = **FIXED** (default)
 - **MAXRECSIZE** = **1**
 - **FRAMESIZE** = **8**
 - **EXTMODE** = (any 8-bit representation)
 - **UPDATEFILE** = **FALSE** (default)
 - Do not specify **BLOCKSIZE**
- ◆ Almost always specified
 - **ANYSIZEIO** = **TRUE**
 - **AREAS, AREALENGTH, FLEXIBLE**

Paradigm MCP-4026 8

To declare a byte stream file in an MCP application, you must specify certain attributes with certain values. This is true whether you need to access a stream file that is stored within the MCP file system, or on a remote system.

- **FILESTRUCTURE** must have the value **STREAM**.
- **FILEORGANIZATION** must be **NOTRESTRICTED**. This is the default, and implies a "flat" file. **KEYEDIO**, **KEYEDIOII**, and **RELATIVE** files cannot be stream files.
- **BLOCKSTRUCTURE** must be **FIXED**. This is also the default.
- **MAXRECSIZE** must have the value **1**. This may seem odd, but we will see shortly that this makes a lot of sense, and enables the MCP to treat byte-stream files much the same way they are treated on other systems.
- **FRAMESIZE** must have the value **8**. Along with the **MAXRECSIZE** setting, this implies each record is a single eight-bit byte.
- **EXTMODE** must be set to one of the mnemonic values for an eight-bit character representation. Common values are **ASCII**, **EBCDIC**, and **OCTETSTREAM**.
- **UPDATEFILE** must have the value **FALSE**. This is also the default.
- **BLOCKSIZE** should not be specified at all, as stream files do not have blocks. Any **BLOCKSIZE** specified will be ignored. Any attempt to interrogate **BLOCKSIZE** for a stream file will result in a non-fatal attribute error.

In addition to these required attributes, you almost always want to specify **ANYSIZEIO=TRUE**. I will discuss why on the next slide.

If you are creating a file, you may also want to specify the **AREAS**, **AREASIZE**, and/or **FLEXIBLE** attributes to configure the total size and storage allocation attributes of the file.

Reading/Writing Byte Stream Files

- ◆ Use standard read/write verbs
- ◆ Programs read/write "chunks," not records
- ◆ With **MAXRECSIZE=1**, each byte in a file is randomly addressable
- ◆ With **ANYSIZEIO=TRUE**, programs can read/write up to $2^{20}-1$ bytes at a time
- ◆ Actual amount read/written is returned in:
 - **CURRENTRECORDLENGTH** file attribute
 - Bits [47:20] in Algol **READ/WRITE** result word
 - Bits [47:20] of COBOL-85 **MCPRESULTVALUE**

Paradigm

MCP-4026 9

Once you have a stream file declared, using it is fairly easy. You use the standard open/close and read/write statements in your programming language. You are generally not reading and writing records, however, but "chunks" of data. The formatting of the data you read and write is entirely up to you. All that the MCP tries to do is move the number of bytes you requested to or from the physical file.

Stream files support both sequential and random I/O. The nice thing about having **MAXRECSIZE=1** and **FRAMESIZE=8** is that each byte in the file is randomly addressable.

With **MAXRECSIZE=1** and **FRAMESIZE=8**, MCP applications reading a stream file will by default read one record at a time – one byte. This is not only tedious, it is very inefficient. This is why the **ANYSIZEIO** attribute exists. Setting **ANYSIZEIO** allows an application to read up to $2^{20}-1$ records (bytes) at a time. The MCP takes care of segmenting the logical reads and writes into appropriate physical reads and writes, crossing disk sector and area boundaries as necessary. Thus, you can randomly position to an arbitrary point in the file and read or write an arbitrary number of bytes.

The only time the MCP may return fewer bytes for a read operation than you asked for with **ANYSIZEIO=TRUE** is when you attempt to read beyond end of file. The MCP will return just the amount of data up to EOF. The EOF indication will not be returned to you until the next read. You can determine how many bytes were actually read in a number of ways, including:

- The value of the **CURRENTRECORDLENGTH** file attribute.
- Bits [47:20] from an Algol **READ** result word (this word has the same format as the **STATE** file attribute). This is much more efficient than accessing the **CURRENTRECORDLENGTH** attribute.
- Bits [47:20] from the COBOL-85 **MCPRESULTVALUE** special register (which also has the format of the **STATE** attribute). This is also a very efficient method of determining actual I/O length.

This is a very brief review of byte-stream files and programming for them. There is more information in the *I/O Subsystem Guide* and *File Attributes Programming Reference Manual*. I gave a UNITE talk on stream files in 2001. There is a link to that presentation and some sample files in the References section at the end of this presentation.

Network Shares

- ◆ Specifically, "shared network resources"
 - File system directories/folders
 - Printers, CD-ROM devices, possibly others
- ◆ MCP uses the Server Message Block (SMB) protocol, also known as
 - Windows/LM (LAN Manager) networking
 - CIFS (Common Internet File System)
 - SAMBA (for Unix/Linux systems)
- ◆ Uses NetBIOS over TCP/IP as a transport
 - TCP ports 137, 138, 139
 - MCP **does not** support port 445 (Windows 2000+)

Paradigm

MCP-4026 10

The next issue to discuss is that of network shares. More specifically, these are known as "shared network resources." The most common resource to share is a disk directory or folder, but many systems also allow the sharing of printers and CD-ROM devices, and possibly other objects as well.

To access these shared resources, the MCP uses a protocol known as Server Message Block, or SMB. This was originally developed by IBM as part of their LanManager product. It was picked up and enhanced by Microsoft, and now forms the basis of Windows networking. It is also known as CIFS (Common Internet File System). Most Unix and Linux systems have compatible implementations, the best known of which is SAMBA.

SMB as implemented on the MCP uses the original NetBIOS over TCP/IP transport mechanism. This uses TCP ports 137 (name service), 138 (datagram service), and 139 (session service). Note that the MCP does not support the implementation of SMB that operates over TCP port 445, which Microsoft introduced with Windows 2000.

Naming Network Resources

- ◆ Resources to be shared are defined by the sharing (server) system
- ◆ Each resource is assigned a "share" name
- ◆ Resources are located using UNC names
 - Universal Naming Convention
 - \\<server name>\<share name>\<path>
 - <server name> can be
 - NetBIOS (simple) host name
 - IP address
 - FQDN (fully-qualified domain name)
 - <path> names the file relative to the shared directory

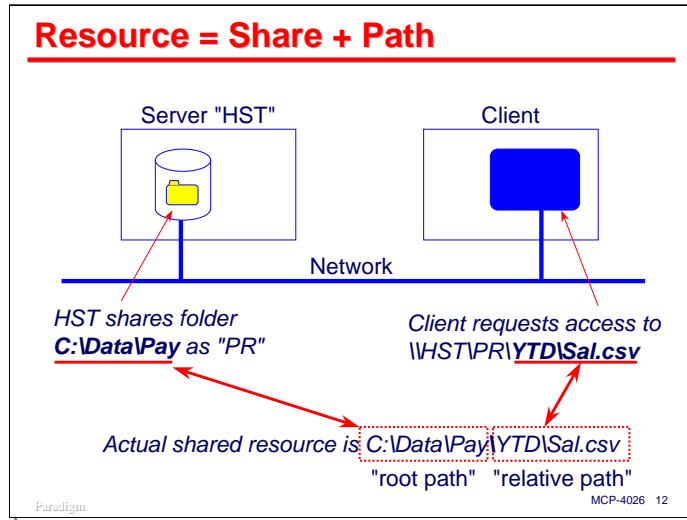
Paradigm

MCP-4026 11

In order to access a remote resource, you need to be able to talk about it. To that end, each resource has a name that is assigned by the sharing (owner, server) system. In SMB, each resource is assigned a "share name," which is unique within that server. There is standard, UNC (Universal Naming Convention), that client systems use to identify a specific resource. The general scheme has two or three parts and is represented as:

\\<server name>\<share name>\<path>

The <server name> is the NetBIOS host name, a FQDN (fully-qualified domain name), or an IP address. The <share name> is the name assigned to the shared resource by the server. The <path> is optional, but almost always used with shared directories. It names a specific file or sub-directory relative to the root of the shared directory.



This diagram illustrates how the naming of shared disk directory resource works. A server system is configured to share a folder "**C:\Data\Pay**" using the share name "**PR**". This is termed the "root folder" or "root path" for the share. The client has no knowledge of where this root folder is located in the server's file system, but constructs a UNC name from the server name ("**HST**"), the share name ("**PR**"), and a relative path. In the example shown on the slide, the relative path consists of a sub-directory and file name within the root directory of the share. The actual file that is accessed is determined on the server by concatenating the root path for the share and the relative path specified in the client's UNC.



With that background, let us now turn to a discussion of the Redirector.

The Redirector

- ◆ Provides access to network shares for MCP applications
- ◆ Companion to Client Access Services
 - CAS provides server-side (inbound) access to MCP files from remote clients
 - Redirector provides client-side (outbound) access to files on remote servers
- ◆ Available since MCP 5.0 (SSR 46.1)
- ◆ Described in *I/O Subsystem Guide*, §29
- ◆ Implemented as a Logical I/O "virtual file"

Paradigm

MCP-4026 14

The purpose of the Redirector is to allow MCP applications access to files on shared directories of remote servers. It functions as a companion to Client Access Services (formerly known as NX/Services).

Client Access Services has been available since the advent of the ClearPath product line. It allows remote client systems to access shared directories in the MCP file system. From the MCP's perspective, it is acting as a server, and this is "inbound" access to MCP files.

The Redirector works in the other direction. It provides "outbound" access from MCP applications (acting as a client) to shared directories on remote systems (which act as servers).

The Redirector has been available since MCP 5.0 (SSR 46.1), ca. 1999. It is documented in the *I/O Subsystem Guide*, "Using the REDIR SUPPORT IOHANDLER Library", currently Section 29.

The Redirector is implemented as a "virtual file" for the MCP's Logical I/O subsystem. What is a virtual file? That is the next topic.

Virtual Files

- ◆ **KIND=VIRTUAL**
 - Semantics of I/O are not implemented by the MCP
 - Open/close, read/write, etc. are implemented by an "I/O Handler" (IOH) library
 - API is described in *I/O Subsystem Guide*, §28
- ◆ Attributes specifically for virtual files:
 - **IOHLIBACCESS**
 - **IOHFUNCTIONNAME**, **IOHTITLE**
 - **IOHINTERFACENAME**, **IOHLIBPARAMETER**
 - **IOHPREFIX**
 - **IOHSTRING**

Paradigm
MCP-4026 15

Virtual files are distinguished by a **KIND** attribute having a value of **VIRTUAL**. They are considerably different from other kinds of MCP files in that the semantics of I/O operations (what actually happens when you use open/close/read/write statements) are not implemented in the MCP. Instead, these semantics are implemented by a special type of library program called an I/O Handler, or IOH.

Unisys has initially implemented two of these virtual file I/O Handlers, the Redirector, and StreamIOH, which we will discuss later in this presentation. You can implement your own I/O Handlers as well. The virtual file API is documented in Section 28 of the *I/O Subsystem Guide*. The late Don Gregory wrote a couple of articles on I/O Handlers in volume 16, number 7 (October/November 2002) of his *ClearPath/A-Series Technical Journal*.

There are several file attributes that are used exclusively with virtual files. Most of these are used to identify and connect with the appropriate I/O Handler library program.

- **IOHLIBACCESS** determines how the library will be linked. It has mnemonic values consistent with the **LIBACCESS** library attribute: **BYFUNCTION** (the default), **BYTITLE**, and **BYINITIATOR**.
- **IOHFUNCTIONNAME** specifies the SL function name for a library if **IOHLIBACCESS=BYFUNCTION**. This is equivalent to the **FUNCTIONNAME** library attribute.
- **IOHTITLE** specifies the library codefile title if **IOHLIBACCESS=BYTITLE**. This is equivalent to the **TITLE** library attribute.
- **IOHINTERFACENAME** specifies the **INTERFACENAME** attribute for the library. This is not used with the Redirector or StreamIOH handlers.
- **IOHLIBPARAMETER** specifies the **LIBPARAMETER** attribute for the library. This also is not used with Redirector or StreamIOH.
- **IOHPREFIX** specifies the prefix string used with the entry points into the library. This is not commonly used.
- **IOHSTRING** is a string-valued attribute that is passed to the library. It is typically used to pass parameters or configuration information from the program declaring the virtual file to the I/O Handler. The format and content of this string is determined by individual handlers. As we will see, this attribute plays an important role in the use of both the Redirector and StreamIOH handlers.

Redirector Implementation

- ◆ **SL REDIRSUPPORT**
- ◆ Library implements SMB protocol for file and printer shares
- ◆ Accessed through the standard Logical I/O API in MCP applications
 - File declarations and attributes
 - Open/close verbs
 - Read/write/seek verbs

Paradigm

MCP-4026 16

As I mentioned, the Redirector is an implementation of a virtual file I/O Handler. It has the default SL function name "**REDIRSUPPORT**" and default codefile name ***SYSTEM/NXSERVICES/REDIRECTOR**. You link to it using **IOHLIBACCESS=BYFUNCTION** (which need not be specified, since that is the default value) and **IOHFUNCTIONNAME="REDIRSUPPORT"**.

The Redirector IOH implements the SMB protocol necessary to access shared file directories on remote systems. It also supports access to shared network printers, and can be used by the Print System to direct printer output from MCP applications to shared printers. I will discuss shared printers briefly later in the presentation.

You access the Redirector using the standard MCP Logical I/O API in your programs. You declare files in the normal way, apply file equation, and use open/close, read/write, and seek verbs as you would for other MCP files. Well, almost – a few of the attributes are different, and the semantics of I/O are somewhat different since you must access the remote files as byte-stream files.

Declaring a Redirector File

- ◆ **KIND=VIRTUAL**,
IOHLIBACCESS=BYFUNCTION,
IOHFUNCTIONNAME="REDIRSUPPORT"
- ◆ Alternatively, set **REDIRECTION=TRUE**
- ◆ Must be a byte-stream file
 - **FILESTRUCTURE=STREAM**, **MAXRECSIZE=1**, etc.
 - Defaults to **EXTMODE=ASCII**
- ◆ **IOHSTRING** attribute
 - Passes parameters to the library
 - Most parameters can be specified by alternate means

Paradigm

MCP-4026 17

When declaring a file for use with the Redirector, you must specify **KIND=VIRTUAL**, **IOHLIBACCESS=BYFUNCTION** (the default), and **IOHFUNCTIONNAME="REDIRSUPPORT"**. Alternatively, you can set all three of these attributes implicitly by setting the file attribute **REDIRECTION** to **TRUE**. **REDIRECTION** is simply a convenience attribute that makes it easy to access the Redirector. You can specify these attributes in a program's file declaration, through file equation, or dynamically in the program at run time, before the file is opened.

Once again, when accessing files on network shares directly, the MCP application must deal with them as byte-stream files. Therefore, the logical file must have the **FILESTRUCTURE**, **MAXRECSIZE**, and other attribute values required for byte-stream files discussed earlier in this presentation. Files using the Redirector default to **EXTMODE=ASCII**, which is generally what you want to enable ASCII/EBCDIC translation between the file on the network share and the MCP application. If you will be processing binary data, you can suppress translation by setting **INTMODE** and **EXTMODE** to the same mnemonic value (**OCTETSTRING** is a good choice) or by setting **DEPENDENTINTMODE=TRUE**.

You will almost always need to specify the **IOHSTRING** attribute for Redirector files. As mentioned earlier, this string-value attribute passes file-specific parameter information to the IOH library. Although **IOHSTRING** is the most common way to pass these parameters, it is possible to convey most parameter information to the Redirector by a couple of alternate means. I will discuss the **IOHSTRING** parameters, and those alternate means, next.

IOHSTRING Parameters	
◆ <u>SERVERNAME</u> =	<i>simple name</i>
◆ <u>SHARENAME</u> =	<i>simple name</i>
◆ <u>DOMAINNAME</u> =	<i>fully qualified name</i>
◆ <u>IPADDRESS</u> =	<i>n.n.n.n</i>
◆ <u>CREDENTIALS</u> =	<i>username/password</i>
◆ <u>USERDOMAIN</u> =	<i>simple name</i>
◆ <u>TIMEOUT</u> =	<i>n [seconds]</i>
◆ <u>SMBTRACE</u> =	TRUE FALSE

Paradigm MCP-4026 18

When used with Redirector files, **IOHSTRING** parameters are written as a series of name=value pairs, delimited by one or more spaces. The parameters may be specified in any order. Some parameters have abbreviations, as shown by underlining on the slide.

- **SERVERNAME** is the NetBIOS name of the remote server hosting the shared directory. Whether this parameter is included in **IOHSTRING** or not, its value is required to make the connection to the remote server.
- **SHARENAME** is the name of the shared resource on the remote server. As with **SERVERNAME**, its value must be specified, whether in **IOHSTRING** or by some other means.
- **DOMAINNAME** is the FQDN (fully-qualified domain name) of the remote server (e.g., mysvr.corp.com). This parameter is optional, but may be necessary if the network cannot resolve the **SERVERNAME** by itself. This may happen if NetBIOS name resolution is not properly configured on the network, or it is blocked by a router or firewall.
- **IPADDRESS** is the Internet Protocol address of the remote server, written in the traditional dotted-decimal notation (e.g., 192.168.16.34). This is also optional, and is typically used only when neither **SERVERNAME** nor **DOMAINNAME** resolution is successful.
- **CREDENTIALS** specifies the username and/or password needed to access the remote resource. As we will see shortly, there is an alternate way to specify credentials that is usually preferable.
- **USERDOMAIN** specifies the user domain or workgroup under which the credentials will be authenticated. On a Windows network, this corresponds to the name of the Windows or Active Directory domain. This parameter is optional, but may be necessary if more than one user domain is visible on the network.
- **TIMEOUT** specifies the number of seconds the Redirector will wait for responses from the remote server before declaring an error. Values between 10 and 60 are usually adequate.
- **SMBTRACE** is a Boolean option that enables diagnostic tracing. The trace data is written to a data file. See UCF 80558376 on <http://support.unisys.com> for more information about the use of this option.

Naming the Shared Resource

- ◆ Analog to UNC `\\server\share\path`
- ◆ Relative path method
 - Specify **SERVERNAME** and **SHARENAME** in **IOHSTRING**
 - Specify the path in one of the file name attributes
 - **FILENAME**, **LFILENAME**
 - **TITLE**, **LTITLE** (*do not use an on-part*)
 - **PATHNAME**
- ◆ Absolute path method
 - **LFILENAME**=***UNC**/`server/share/path`
 - Overrides server or share specified in **IOHSTRING**
 - ***UNC** prefix is understood only by the Redirector

Paradigm

MCP-4026 19

In addition to providing parameters through **IOHSTRING** or one of the alternative methods, you must identify the specific file or sub-directory within the network share that you wish to access. Redirector provides two ways to do this, both analogous to the UNC "`\\<server name>\<share name>\<path>`" notation.

The first method uses a relative path name. You specify the **SERVERNAME** and **SHARENAME** parameters in **IOHSTRING**. You then specify the `<path>` portion of the UNC name through one of the MCP file name attributes, **FILENAME**, **LFILENAME**, **TITLE**, **LTITLE**, or the POSIX-oriented **PATHNAME** attribute. It is generally a good idea to use the "L" attributes, as the **FILENAME** and **TITLE** attributes will truncate node names longer than 17 characters. Also, do not use an on-part in **TITLE** or **LTITLE**. Doing so implicitly changes the **KIND** of the file to **DISK**, at which point it is no longer a virtual file, and thus will not use the Redirector. The MCP will simply look for the file name in its local file system.

The second method uses an absolute path name that is very similar to the standard UNC notation. You specify one of the file name attributes with a first node of "***UNC**". The second node must be the `<server name>` and the third node must be the `<share name>`. The remaining nodes in the file name specify the relative `<path>` name to the specific file or sub-directory you are trying to access. Any server or share name you specify using this file name notation overrides any server or share name present in **IOHSTRING**. Note that MCP file name attributes require you to use forward slashes (/), not the backslashes (\) typically used in UNC notation. As with the relative path method, do not include an on-part when using the **TITLE** or **LTITLE** attributes. Finally, note that the "***UNC**" prefix is recognized only by the Redirector. Used with any other type of file, it is taken literally as part of the file name.

Example: Algol with Relative Path

```
FILE SALFILE (REDIRECTION,  
  FILESTRUCTURE=STREAM,  
  MAXRECSIZE=1,  
  FRAMESIZE=8,  
  ANYSIZEIO,  
  EXTMODE=ASCII,  
  INTMODE=EBCDIC,  
  FLEXIBLE,  
  FILEUSE=IO,  
  LTITLE="YTD/""Sal.csv"".",  
  IOHSTRING="SERVER=HST SHARE=PR TIMEOUT=30"  
    " CREDENTIALS=USAH/PW"  
    " USERDOMAIN=MYCORP"  
    " IPADDRESS=192.168.54.211");
```

Paradigm

MCP-4026 20

This slide shows an example of how you might declare a file in Algol for use with the Redirector. Note the use of the **REDIRECTION** attribute to set the **KIND**, **IOHLIBACCESS**, and **IOHFUNCTIONNAME** attributes to connect with the Redirector. The next several attributes configure the file to access a byte stream. The **LTITLE** attribute designates a path name that will be relative to the server and share names specified in the **IOHSTRING** attributes. In this case the **IOHSTRING** includes the **CREDENTIALS** and **USERDOMAIN** parameters for the remote server. It also includes the IP address, which is not normally something you want to specify unless NetBIOS and FQDN name resolution fails to work.

Example: COBOL with Absolute Path

```
FD SAL-FILE
  RECORD CONTAINS 1 TO 8192 CHARACTERS
                        DEPENDING ON W-BSF-SIZE
  VALUE OF
    REDIRECTION IS TRUE
    FILESTRUCTURE IS STREAM
    MAXRECSIZE IS 1
    BLOCKSTRUCTURE IS FIXED
    FRAMESIZE IS 8
    EXTMODE IS ASCII
    ANYSIZEIO IS TRUE
    LTITLE IS "*UNC/HST/PR/YTD/" "Sal.csv"
    IOHSTRING="CREDENTIALS=USAH/PW
-           "USERDOMAIN=MYCORP TIMEOUT=30
-           "IPADDRESS=192.168.54.211".
```

Paradigm

MCP-4026 21

This slide shows a similar file declaration as it would appear for COBOL-74 or COBOL-85. The main difference is that the file on the remote server is identified using the absolute path method with the ***UNC** file name prefix. Note that the server and share names are part of the **LTITLE** attribute in this example instead of being specified as parameters in the **IOHSTRING** attribute.

Other File Attributes Supported	
◆ DEPENDENTSPECS	◆ FILELENGTH
◆ DEPENDENTINTMODE	◆ LASTRECORD
◆ OVERRIDEEXTMODE	◆ SYNCHRONIZE
◆ ACCESSDATE/TIME	◆ SECURITYMODE
◆ ALTERDATE/TIME	◆ SECURITYTYPE
◆ CREATIONDATE/TIME	◆ SECURITYUSE
◆ FILEKIND=PERMDIR	◆ USERINFO (DOS attribute bitmask)

Paradigm MCP-4026 22

The Redirector supports a number of other file attributes, primarily for interrogating properties of the shared file and to set its security.

- The date/time attributes will return the related timestamps from the remote system, if they are supported by that system.
- The concept of **FILEKIND** is not supported by SMB. You can set and interrogate **FILEKIND** on a Redirector file, but with one exception the value is simply stored in the logical file and not propagated to the remote system. The exception is the value **PERMDIR**, which is used to open and create sub-directories within the shared directory. I will discuss this further shortly.
- **FILELENGTH** and **LASTRECORD** will return the size of the file in bytes once the file is opened.
- **SYNCHRONIZE** can be used to flush buffers when writing to the remote system.
- The security attributes can be used to interrogate access permissions for a file, and when creating a file, to set the permissions on the remote system.
- **USERINFO** is a general-purpose 48-bit word-valued (**REAL** in Algol and COBOL) attribute. With the Redirector, this attribute is used to set and interrogate a bit mask of properties, similar to the "attributes" property in Windows. The bits are:
 - [0:1] Read-only flag
 - [1:1] Hidden file flag
 - [2:1] System file flag
 - [3:1] Volume ID flag (read only, cannot be set)
 - [4:1] Directory flag (read only, cannot be set)
 - [5:1] File changed flag (also known as the "archive bit")

NXCONFIG Files

- ◆ IOHSTRING parameters can be stored in a separate MCP file
- ◆ Uses a variation of the *UNC convention
 - LFILENAME=*NXCONFIG/*configname/path*
 - Parameters are found in a file named **NXSERVICES/CONFIG/*configname***
 - Must have **FILEKIND=SEQDATA**
 - Max 250 chars read (including spaces)
 - "%" trims spaces from config file records
 - Server name and share name are taken from the configuration file parameters
 - Cannot include **TIMEOUT** in NXCONFIG

Paradigm

MCP-4026 23

I have mentioned that the **IOHSTRING** parameters can be specified by alternate means. One of those means involves the use of NXCONFIG files. These are simply small text files that store the parameters in the same format as they would be in an **IOHSTRING** value.

You specify the use of an NXCONFIG file using a variation on the ***UNC** prefix convention. If the first node of the file name is "***NXCONFIG**", the Redirector takes the second node of the file name and prepends that with "**NXSERVICES/CONFIG/**" to form the name of the file from which it will read the parameter data. The third and following nodes then comprise the relative path portion of the UNC name.


NXCONFIG files are searched for using normal usercode and family substitution conventions. They must have a **FILEKIND** of **SEQDATA**. The Redirector will read a maximum of 250 characters from the file, including any trailing spaces in the text portion of the record (250 divided by 72 characters per line is almost 3.5 lines). You can spread the parameter data over more lines by using a "%" at the end of lines to terminate the accumulation of characters on that line.

The **SERVERNAME** and **SHARENAME** parameters should be included in the NXCONFIG file, as this method supports only the relative path name convention.

A simple NXCONFIG file might look like this

```
SERVER=DIGMHPO6 SHARE=TESTSHARE %  
SMBTRACE=FALSE %  
IPADDRESS=192.168.16.2 %
```

User Authentication for the Share

- ◆ **IOHSTRING CREDENTIALS** parameter
 - Supplies a username and password to the server
 - Possibly qualified by the **USERDOMAIN** parameter
 - **Note:** *this exposes the password in clear text* 
- ◆ **Credentials files**
 - Allows credentials for remote servers to be stored in an encrypted MCP file
 - Searched for automatically if **CREDENTIALS** not specified in **IOHSTRING**
 - Can be protected by standard MCP file security (**GUARDFILE**, etc.)

Paradigm MCP-4026 24

Most remote systems will require that you present credentials in the form of a username and password in order to authenticate and control access to the shared resource. One way to do that is by including a **CREDENTIALS** parameter in the **IOHSTRING** attribute. The **USERDOMAIN** parameter is used with **CREDENTIALS** to specify the user domain in which the credentials will be authenticated.

The problem with using the **CREDENTIALS** parameter is that it exposes the password in the clear. This is a potentially serious security issue.

The Redirector supports another method of specifying credentials for a remote system. The remote username and password (and optionally, the name of the user domain) can be encrypted and stored in a small MCP file, called a "credentials file." If the **CREDENTIALS** parameter is not specified in **IOHSTRING** (or a **NXCONFIG** file), then Redirector looks for a credentials file based on the **SERVERNAME** parameter. If this file is found, the credentials from that file are transmitted to the remote system for authentication.

Credentials files are stored under the usercode that is opening the Redirector file. It is also possible to have a "global" credentials file stored under the "*" node of the directory. Such a credentials file can be accessed by multiple usercodes. Credentials files can be protected by standard MCP file security, particularly by the use of guard files.

Creating Credentials Files

- ◆ Created by special MCP utility
 - ***SYSTEM/NXSERVICES/MAKECREDENTIALS**
 - Single space-delimited string parameter containing
 - Remote server host name or "*"
 - User name
 - Password (may be in quotes to preserve case)
 - Authentication domain name (optional)

- ◆ Generates a small file under the originating MCP usercode
 - **NXSERVICES/CREDENTIALS/<server name>**
 - **NXSERVICES/CREDENTIALS** (if server name="*")

Paradigm MCP-4026 25

Credentials files can be created by means of a utility program that is part of the standard software release, ***SYSTEM/NXSERVICES/MAKECREDENTIALS**. This program takes a single string parameter. The parameter string must contain three or four space-delimited tokens:

- The first token is the name of the remote server with which the credentials file will be used. If this token is an asterisk (*), then the utility generates a so-called "default" credentials file than can be used with any server within a user domain.
- The second token is the username to be authenticated on the remote system. This is generally case-insensitive.
- The third token is the password for that username. The case sensitivity of the password varies, depending on the version of the MCP and the type and version of the remote system. In general, the more recent the version of either MCP or Windows software involved, the more likely it is that the password will be case sensitive. This token may be embedded in double quotes (") if it contains lower-case and/or non-alphanumeric characters.
- The fourth token is the name of the user domain. This is optional.

The utility encrypts the parameters and stores them in a file, named

NXSERVICES/CREDENTIALS/<server name>

under the initiating user's usercode. If the parameter had an asterisk as the first token, the credentials file will be named simply **NXSERVICES/CREDENTIALS**. When opening a file, the Redirector will search first for a credentials file for the specific server being accessed. If no such credentials file exists, it will then search for a default credentials file.

Credentials File Examples

- ◆ WFL/CANDE RUN command

```
RUN *SYSTEM/NXSERVICES/MAKECREDENTIALS  
  ("WINSERV PAUL "my!%#@&PW" MYCORP")
```
- ◆ CANDE UTILITY command

```
U *SYSTEM/NXSERVICES/MAKECREDENTIALS  
  WINSERV paul "myPW" MYCORP
```
- ◆ Creating a "default" credentials file

```
RUN *SYSTEM/NXSERVICES/MAKECREDENTIALS  
  (** PrintServ "ps-pw" MYCORP)
```

Paradigm

MCP-4026 26

This slide shows three examples of credentials file generation using **MAKECREDENTIALS**. Note the use of doubled double-quotes within the quoted parameter strings. The CANDE **UTILITY (U)** command is an especially nice way to generate credentials, as it eliminates the need to quote the parameter string and supply extra quotes for quoted passwords.

Credentials File Conventions

- ◆ Can be used only from the usercode that created it
- ◆ A non-usercoded run of **MAKECREDENTIALS**
 - Creates a "global" credentials file
 - Usable from any usercode
- ◆ Search order (family substitution applies)
 1. **CREDENTIALS** in **IOHSTRING**
 2. (*usercode*)**NXSERVICES** / **CREDENTIALS** / *server*
 3. (*usercode*)**NXSERVICES** / **CREDENTIALS**
 4. ***NXSERVICES** / **CREDENTIALS** / *server*
 5. ***NXSERVICES** / **CREDENTIALS**

Paradigm MCP-4026 27

There are some important conventions and restrictions that apply to the use of credentials files.

The most significant restriction is that a credentials file may only be used from the usercode that created it. The **MAKECREDENTIALS** utility stores the creating MCP usercode in the file along with the authentication data you specified. When the Redirector attempts to use a credentials file, it checks that the usercode directory under which the file was found matches the usercode stored in the file. If these do not match, the connection to the remote server is terminated with an open error. Thus, you cannot copy someone else's credentials file to your usercode and use it.

It is possible, however, to create a so-called "global" credentials file that can be used from any MCP usercode. To do this, you must run **MAKECREDENTIALS** without a usercode (e.g., from the ODT). The file will be stored under the ***NXSERVICES** directory.

As the slide shows, credentials are searched for in priority order. If the **CREDENTIALS** parameter is present in **IOHSTRING** or an **NXCONFIG** file, those are used regardless of the presence of any credentials files. If a **CREDENTIALS** parameter is not present, the Redirector searches for a credentials file first under the opening task's usercode and for the specific server being accessed, cascading down through the combinations of default and global credentials files. Normal family substitution applies when searching for credentials files.

Redirector Directory Operations

- ◆ Directories within network shares are a lot like MCP permanent directories
 - Are physically present, not just a node in a name
 - Must be explicitly and individually created and deleted
 - Cannot have a file and a directory of the same name
- ◆ Directory operations
 - Specifying **FILEKIND=PERMDIR** is required
 - Create: Open with **NEWFILE=TRUE**
 - Delete: Open, then close with purge
 - Rename: Open, then change **LFILENAME**
 - Read: Returns a stream containing file entries
Format defined in *I/O Subsystem Guide*

Paradigm

MCP-4026 28

Directories or folders on most other systems are not like the "virtual directories" we use in the MCP environment. Directories on other systems are physically present, and are often physically represented as a file, not just a node in a multi-level file name. These must be explicitly and individually created and deleted, perhaps using commands similar to **MKDIR** and **RMDIR**. Because the directories are stored in much the same way a file is, most systems do not support the MCP's ability to have a directory and a file with the same name.

The Redirector supports a basic set of operations against directories on network shares. To access a directory, you open the directory through Redirector as if it were a file, but you must explicitly set **FILEKIND=PERMDIR**. This attribute must be set in the file's source code declaration or dynamically at run time – it cannot be established through file equation. You can perform the following operations on these directory "files":

- **Create** – setting **NEWFILE=TRUE** when opening a Redirector file with **FILEKIND=PERMDIR** will create a new, empty directory on the remote share. If a file or directory of that name already exists, an open error occurs. Any higher-level sub-directories must already exist, or an open error occurs as well.
- **Delete** – opening the directory with **NEWFILE=FALSE** and then closing the file with the **PURGE** option will delete the directory on the remote share.
- **Rename** – opening the directory as input and then changing the **FILENAME** or **LFILENAME** attribute will change the name of the directory on the remote share.
- **Read** – Opening the directory as input and reading as if it were a file returns a stream of the subordinate file and sub-directory entries. These entries are prefixed with a binary length field and contain a mixture of binary and EBCDIC data. The format of these directory entries is documented in Section 29 of the *I/O Subsystem Guide*.

Writing to a directory file is not allowed. Directories are modified only by creating, deleting, and renaming the subordinate files and sub-directories they contain. Changing the **FILEKIND** to or from **PERMDIR** on an open Redirector file is also not allowed.

Redirector Tips

- ◆ User authentication is often a problem area
 - Beware of password case sensitivity
 - Case sensitivity varies by Windows and MCP version
 - When in doubt, use matching case
- ◆ NetBIOS name resolution is also a problem
 - Try using **DOMAINNAME=<FQDN>**
 - If necessary, use **IOHSTRING IPADDRESS**
- ◆ Subdirectories on the share must exist – not created automatically
- ◆ Beware of ASCII/EBCDIC translation

Paradigm

MCP-4026 29

Here are some tips on the use of the Redirector that I've gleaned from my experience with it.

First, if you are going to have a problem accessing a shared resource, the problem is probably going to involve either user authentication or name resolution. Most authentication problems I have seen result from mismatched expectations on the case sensitivity of passwords. When in doubt, try to match case exactly.

NetBIOS server name resolution is also a common problem, especially on networks with incompletely or improperly configured domain controllers or DNS servers. Sometimes supplying a fully-qualified domain name with the **DOMAINNAME** parameter will get around this. As a last resort use the **IPADDRESS** parameter, but beware that this may cause a maintenance problem down the road, and may not work at all for servers that have dynamically-assigned addresses.

As MCP users, we are very used to the idea that sub-directories in file names come and go automatically as needed. That is not the case with sub-directories on most other types of systems. For Windows and Unix/Linux systems especially, sub-directories must be explicitly created and deleted. Attempting to create a file without having all of the sub-directories in its path present results in an open error. The Redirector will not create these for you automatically. You must first create the necessary sub-directories on the remote system or use the directory manipulation facilities of the Redirector. Obviously, the user account you are authenticating with the remote share must have appropriate rights to any directories and files you will be accessing.

Finally, beware that the Redirector by default uses **EXTMODE=ASCII** and most MCP application by default have **INTMODE=EBCDIC**. This will result in ASCII/EBCDIC translation for I/Os done through the Redirector. For text files, this is usually what you want; for binary data, this is usually disastrous. You can disable translation by setting **INTMODE** and **EXTMODE** to the same value, or by setting the **DEPENDENTINTMODE** attribute to **TRUE**.

Defining Printer Shares

- ◆ See "Setting up a Connection to a Network Printer Share" in the *Installing a Printer for MCP Print System Use* manual

- ◆ Example:

```
PS CONFIG +VIRTUAL SHAREPRN
BLOCKSIZE=10000,
TRANSLATION=NONE,
PROTOCOL=TRANSPARENT,
PPT=NONE, SPOOLER=NONE,
IOH="NXPRINT (SERVER=PSVR1 SHARE=LAZR2
      NXCREDS=PUSSR TIMEOUT=30)
      IN SL REDIR SUPPORT"
DRIVER=..., etc.
```

Paradigm

MCP-4026 30

I mentioned earlier that the ***SYSTEM/NXSERVICES/REDIRECTOR** library can also serve as an I/O Handler for the Print System. This allows you to configure printers in the Print System that route their output to a shared network printer. The current library replaces the older I/O Handler for network printing, ***SYSTEM/NXSERVICES/PRINT/REDIRECTOR**, that was available prior to MCP 5.0.

A detailed description for configuring shared network printers can be found in the section "Setting up a Connection to a Network Printer Share" in the *Installing a Printer for MCP System Use* manual. The slide shows a basic example of such a printer configuration. The parameters in parentheses for the **NXPRINT** IOH are similar to those for the IOHSTRING file attribute. **NXPRINT** supports credentials files and a template mechanism that eases the task of configuring large numbers of network printers. Global and default credentials files are especially useful, as a single credentials file can permit access to a large number of printers by a large number of Print System users.

PCDRIVER Utility

- ◆ ***SYSTEM/NXSERVICES/PCDRIVER**
 - Takes a single string parameter
 - Documented in *System Software Utilities Operations Guide*

- ◆ **Capabilities**
 - Interfaces to the Windows-based "Launcher" utility
 - Transfers files between MCP and an SMB share
 - Binary file transfer
 - Text file transfer with MCP record format mapping
 - Removes files from an SMB share
 - Does not overwrite existing files (MCP or share)
 - Stops on first error; reports in **TASKVALUE. [5 : 6]**

Paradigm MCP-4026 31

In the beginning of this presentation I made the point that the Redirector is a file *access* mechanism rather than a file *transfer* mechanism. That does not mean that it cannot be used for file transfer, however. It is quite easy with the Redirector to write a program that copies a stream file between a network share and the MCP file system.

Unisys supplies a utility program as part of the standard software release, ***SYSTEM/NXSERVICES/PCDRIVER**, that handles a variety of common file transfer tasks for SMB shared directories. This program is documented in the *System Software Utilities Operations Guide*.

PCDRIVER has two main modes of operation. One is to communicate with a Windows-based program, **Launcher** (also part of the standard release and available from the **Installs** share) over a standard TCP/IP port. This mode does not involve the SMB protocol, and I will not discuss it further in this presentation.

The second mode of operation uses the Redirector and the SMB protocol to access files on remote shares. In this mode, **PCDRIVER** supports three primary operations:

- Transfer binary byte-stream file between a remote share and the MCP.
- Transfer a text file between a remote share and the MCP, converting between line-oriented text format on the remote share and record-oriented format on the MCP.
- Remove a file from an SMB share. Only individual files can be removed by **PCDRIVER**, not whole directories.

PCDRIVER will not allow you to overwrite an existing file on either the MCP or the shared directory. You must explicitly remove existing files before transferring a file of the same name to either type of destination.

PCDRIVER allows you to submit multiple commands in one run. It will stop processing commands on the first error it encounters. It returns an error code in the low-order six bits of its **TASKVALUE** task attribute. These error codes are documented in the *Operations Guide*.

PCDRIVER Parameter

- ◆ In-line syntax
 - Remote server specification
 - IP address or FQDN (for Launcher interface)
 - \\servername\sharename (for SMB interface)
 - [server options] (optional)
 - List of commands separated by ";"
- ◆ Command-file syntax
 - RUN .../PCDRIVER ("FILE MY/PCD/PARAMS");
 - Reads parameter data from the named file
 - Must be an MCP FILEKIND of type text
 - File contents are same format as in-line syntax

Paradigm MCP-4026 32

PCDRIVER is controlled by a single string parameter. The string can contain either the commands directly, or it can contain reference to an MCP text file where the commands are stored. If the string parameter begins with the word "**FILE**", then the commands are taken from the file having the title that follows. Otherwise, **PCDRIVER** expects to find its commands in the string parameter.

The command syntax begins with a designation of the destination and the mode in which **PCDRIVER** will run:

- If the command syntax starts with an IP address or a fully-qualified domain name (e.g., server.company.com), then **PCDRIVER** opens a connection to the Launcher program on the specified system.
- If the command syntax starts with a server name and share name in UNC format (including backslashes, e.g., \\myserver\myshare), **PCDRIVER** uses Redirector to communicate with that network share.

Following this initial designation of the mode and destination, the syntax may optionally include a set of options in square brackets that apply to the entire run. Some of these options configure the Redirector; the rest supply defaults for converting between line-oriented and record-oriented file formats.

The remainder of the command syntax consists of **PCDRIVER** commands. If there are multiple commands, they must be delimited by semicolons (;).

If you direct **PCDRIVER** to read its commands from an external file, the file must have one of the text-oriented **FILEKINDs** – **TEXTDATA**, **SEQDATA**, **JOBSYMBOL**, etc.

PCDRIVER SMB Server Options

- ◆ **CREDENTIALS**
 - Username/password
 - #*nodename* (to specify alternate CREDENTIALS file)
 - Attempts to use a CREDENTIALS file if not specified
- ◆ **USERDOMAIN = *simple name***
- ◆ **NODISPLAY**
- ◆ **<text data option>**
 - Multiple text options can be specified
 - These are defaults
 - Can be overridden by text options on individual file-transfer commands

Paradigm MCP-4026 33

Server options are optional in a **PCDRIVER** command string, but if present, must be coded within square brackets following the "\\servername\sharename" syntax. Multiple options must be separated by commas. The options shown here are just the ones for the Redirector mode of operation. There are additional options used with the Launcher mode.

- **CREDENTIALS** – this option specifies user credentials similar to the parameter with the same name in a Redirector **IOHSTRING** attribute. Instead of a username/password pair, you can also specify a single filename node, prefixed by a number sign (#). Using this form will attempt to locate a credentials file with the specified name as the last node in its file name. If **CREDENTIALS** is not specified, the Redirector will look for a credentials file in the manner described earlier for Redirector files.
- **USERDOMAIN** – this option is identical to the one of the same name in a Redirector **IOHSTRING** attribute.
- **NODISPLAY** – specifying this option causes **PCDRIVER** to suppress the display of error messages that indicate a command failure. Regardless of the presence of this option, **PCDRIVER** stops executing commands upon the first failure, and stores an error code in the low-order six bits of its **TASKVALUE** attribute.
- <text data option> – this is one or more of the options for mapping between line-oriented text files on the remote share and a record-oriented MCP file. Options specified here serve as defaults for any text-oriented file transfer commands that follow. These options may be overridden by <text data option> specifications on individual commands. These options will be discussed shortly.

PCDRIVER SMB Commands

- ◆ **BINARYDATATOPC**
 <MCP title> <PC path>
- ◆ **BINARYDATAFROMPC**
 <MCP title> <PC path>
- ◆ **TEXTDATATOPC**
 [<text data options>] <MCP title> <PC path>
- ◆ **TEXTDATAFROMPC**
 [<text data options>] <MCP title> <PC path>
- ◆ **REMOVE** <PC path>

Paradigm MCP-4026 34

There are five commands that can be used with the Redirector mode of **PCDRIVER**.

- **BINARYDATATOPC** – Transfer a data file from the MCP file system to the remote network share. No translation or record formatting takes place. The MCP file does not need to be a stream file, but does need to have **BLOCKSTRUCTURE=FIXED** and **FRAMESIZE≠4**. Records are written to the remote share head-to-tail, with no line delimiters. The command will fail if a file with the same name as <PC path> is present on the remote share.
- **BINARYDATAFROMPC** – Transfer a data file from the remote network share to the MCP file system. No translation or record formatting takes place. The file created on the MCP will be a stream file with **EXTMODE=ASCII**. The command will fail if a file with the same name as <MCP title> is present in the MCP file system.
- **TEXTDATATOPC** – Transfer a file from the MCP file system to the remote network share. If the MCP file is encoded in EBCDIC, the data will be translated to ASCII according to the **CCSVERSION** text data option, if any. The MCP file does not need to be a stream file, but does need to have **BLOCKSTRUCTURE=FIXED** and **FRAMESIZE≠4**. Records are written to the remote share with trailing blanks trimmed and a carriage-return/line-feed pair inserted after each trimmed record. The command will fail if a file with the same name as <PC path> is present on the remote share.
- **TEXTDATAFROMPC** – Transfer a file from the remote network share to the MCP file system. Unless the **TRANSLATE** option is **FALSE**, the data will be translated from ASCII to EBCDIC according to the **CCSVERSION** option, if there is one. The file created on the MCP will be a record-oriented file with **EXTMODE=EBCDIC**. The command will fail if a file with the same name as <MCP title> is present in the MCP file system.
- **REMOVE** – Removes the named file from the remote network share. If no such file is present on the share, the command is ignored and no error is generated. If the file is a directory, the command fails.

Note that **TEXTDATATOPC** and **TEXTDATAFROMPC** can both accept <text data options> if encoded in square brackets after the command name. These options are discussed next.

PCDRIVER Text Data Options	
Option syntax (defaults underlined>	to/from
CCSVERSION <version> (<i>defaults to SYSOPS</i>)	both
SEQUENCENUMBERSONPC = <u>TRUE</u> <u>FALSE</u>	from
FILEKIND <filekind> DATA n	from
BLOCKING = n (<i>default depends on rec size</i>)	from
UNITS <u>CHARACTERS</u> WORDS	from
AREABYTES = n (<i>default 81,000</i>)	from
RECORDS <u>CRLF</u> IMPLICIT	from
OVERFLOW = <u>TRUNCATE</u> FOLD WRAP ERROR	from
TRANSLATE = <u>TRUE</u> FALSE	from
TRIMBLANKS = <u>TRUE</u> FALSE	to

Paradigm MCP-4026 35

The <text data options> portion of the **PCDRIVER** command syntax controls how data is mapped between line-oriented stream files on the remote network share and record-oriented files in the MCP file system. These options can be specified only for the **TEXTDATATOPC** and **TEXTDATAFROMPC** commands. They can also be specified as global options at the beginning of the command string, but with one exception discussed below, apply only to the **TEXTDATATOPC** and **TEXTDATAFROMPC** commands. These options are always written as a comma-separated list within square brackets.

- **CCSVERSION** specifies which variant of ASCII/EBCDIC translation will take place. This defaults to the system-level setting determined by the ODT **SYSOPS CCSVERSION** command.
- **SEQUENCENUMBERSONPC** indicates whether the line-oriented file on the remote network share has (or should have) sequence numbers in the position determined by the file's **FILEKIND** attribute. The default is **FALSE**.
- **FILEKIND** applies only to the **TEXTDATAFROMPC** command. It specifies the **FILEKIND** mnemonic that will be assigned to the MCP file that is created by the transfer operation. In addition to the standard **FILEKIND** mnemonics, you can specify "DATA n" to indicate a data file with n-character records. The default size for data file records is 90 characters.
- **BLOCKING** specifies the blocking factor for MCP files, and is valid only for the **TEXTDATAFROMPC** command.
- **UNITS** specifies the value of the **UNITS** file attribute for the MCP file, and is valid only for the **TEXTDATAFROMPC** command.
- **AREABYTES** specifies the size of disk areas for the MCP file and is valid only for the **TEXTDATAFROMPC** command. Also applies to the **BINARYDATAFROMPC** command when specified as a global default text option.
- **RECORDS** specifies whether record boundaries in the line-oriented file on the network share should be determined from the presence of line delimiters (**CRLF** mnemonic, the default) or fixed character intervals (**IMPLICIT** mnemonic) determined by the **FILEKIND** and **SEQUENCENUMBERSONPC** options. This option is valid only for the **TEXTDATAFROMPC** command.
- **OVERFLOW** specifies how line-oriented records that exceed the size of the MCP record area are to be handled. Valid mnemonic values are **TRUNCATE** (the default), **FOLD** (using "\ " as a fold character), **WRAP** (as in word wrap), or **ERROR**. This option is valid only for the **TEXTDATAFROMPC** command, and ignored when **RECORDS=IMPLICIT**.
- **TRANSLATE** specifies whether ASCII/EBCDIC translation will occur. This option is valid only for the **TEXTDATAFROMPC** command. The default value is **TRUE**.
- **TRIMBLANKS** specifies whether trailing blanks are trimmed from lines written to the remote share. This option is valid only for the **TEXTDATATOPC** command.

PCDRIVER Example

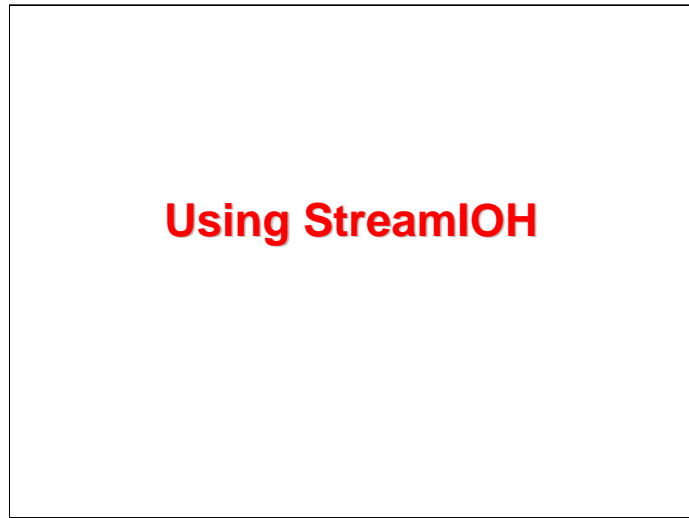
```
RUN *SYSTEM/NXSERVICES/PCDRIVER(  
  "\\HST\PAY [AREABYTES=900000] " &  
  
  "BINARYDATATOPC MY/BLOB blobs\my.dat;" &  
  
  "TEXTDATAFROMPC [" &  
    "FILEKIND DATA 360," &  
    "UNITS=WORDS," &  
    "BLOCKING=30," &  
    "OVERFLOW=ERROR] " &  
    "DATA/FROM/PC ON DROP toMCP.txt;" &  
  
  "REMOVE toMCP.txt");
```

Paradigm

MCP-4026 36

This slide shows a sample run of the **PCDRIVER** program. The command string breaks down as follows:

- `\\HST\PAY` specifies that the program will use its Redirector mode to connect to the remote system. The server name is **HST** and the share name is **PAY**.
- `[AREABYTES=900000]` specifies that MCP files created by this run will have an **AREALENGTH** of 900,000 bytes.
- The **BINARYDATATOPC** command will transfer the MCP file "**MY/BLOB**" to the network share sub-directory "**blobs**" and file name "**my.dat**" without translation or conversion to line-oriented format.
- The **TEXTDATATOPC** command will transfer the "**toMCP.txt**" file from the network share to the MCP file titled "**DATA/FROM/PC ON DROP**". The data will be translated from ASCII to EBCDIC (**TRANSLATE=TRUE** by default). The MCP file that is created will have **FILEKIND=DATA**, **UNITS=WORDS**, **MAXRECSIZE=60** (360 bytes÷6 bytes/word), and **BLOCKSIZE=1800** (60×30). With **OVERFLOW=ONERROR**, if any lines from the file on the network share are longer than 360 characters, an error will be generated for the command.
- The **REMOVE** command will remove the file just transferred from the network share. Note that this command will not be executed if either of the preceding commands fails.



That concludes our discussion, for the time being, of the Redirector. One of the issues with the Redirector is that MCP applications using it must access files on network shares as byte-stream files. If the file has binary data, or the program is planning to parse the data in some way, this may be appropriate.

What happens though, if the file on the share is line-oriented text and you simply want to read the lines as if they were traditional records? That would generally involve reading chunks of data from the stream file, parsing the data for line delimiters, assembling whole lines (which could be split across chunks) into records, and then doing the necessary record-oriented processing. Such a task is not rocket science, but it's not trivial, either, especially in a language like COBOL.

This is where StreamIOH comes in.

What is StreamIOH?

- ◆ Maps I/Os between traditional MCP record formats and line-oriented text file formats
 - Allows an MCP application to read and write byte stream files *as if they were record files*
 - Mapping to/from record format is done on the fly
- ◆ Another virtual file implementation
 - **SL STREAMIOH**, available since MCP 7.0
 - Use with MCP byte stream files
 - Use with files on remote servers via Redirector
 - *Invoked solely with file attributes!*
 - No program changes necessary
 - Can be implemented using only file equation

Paradigm

MCP-4026 38

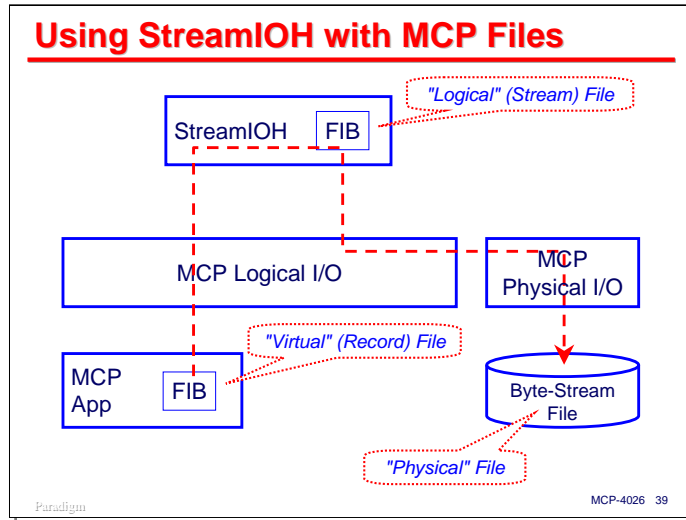
StreamIOH is a facility that can map logical I/Os between the record-oriented formats we traditionally use with MCP applications and line-oriented text file formats typically used on other systems. It allows MCP applications to read and write line-oriented text files as if they were record files. This conversion is generally done on the fly, as records are being read and written. With one exception that we will discuss later, the file is not converted en masse between line-oriented and record format in a separate pass.

Like the Redirector, StreamIOH is virtual file implementation. By default, it is accessed through the SL function name **STREAMIOH**. It has been available since MCP 7.0 (SSR 48.1), ca. 2002, but its implementation was incomplete until MCP 9.0. It is bundled in the standard system software release.

You can use StreamIOH in two ways:

- To read and write line-oriented byte stream files within the MCP file system.
- In conjunction with the Redirector, to read and write line-oriented byte stream files on a network share.

One of the really nice things about StreamIOH is that can be invoked solely with file attributes. This means that you can give an existing program the ability to read and write variable-length, line-oriented text files without changing the program or even recompiling it. StreamIOH can be implemented simply using file equation.



This diagram shows how StreamIOH works when accessing byte stream files within the MCP file system. Since it is implemented as a virtual file, MCP applications access it through a standard file declaration and I/O verbs in the programming language. The data structure that represents a file at run time is called a File Information Block, or FIB. The MCP's Logical I/O module connects to the StreamIOH library to process the MCP application's I/O requests. The result of these requests will be viewed by the application as full records.

The StreamIOH library dynamically creates another FIB to access the physical byte stream file. The library reads or writes chunks of stream data for the physical file as necessary, and translates between the line-oriented text format of the physical file and the record-oriented format that the MCP application sees.

StreamIOH Example and Demo

```
RUN MY/PROG; FILE FIXED (  
  KIND=VIRTUAL, IOHFUNCTIONNAME="STREAMIOH",  
  LFILENAME=MY/STREAM.TXT, FAMILYNAME=PACK);
```

Paradigm

MCP-4026 40

To see how easy it is to use StreamIOH, this slide shows a very basic example. Assume that the program **MY/PROG** has been written to read from the file with **INTNAME "FIXED"** in the traditional record-oriented manner. Assume also that we have a line-oriented text file with carriage-return/line-feed delimiters stored on the **PACK** family as **MY/ "STREAM.TXT"**.

To allow the program to read this stream file in a record-oriented fashion, all we need to do is file equate **KIND=VIRTUAL** and **IOHFUNCTIONNAME="STREAMIOH"**. There are a number of additional options that can be specified, but the standard defaults allow us to read a simple line-oriented text file without specifying anything else.

[Demo here]

Declaring a StreamIOH File

- ◆ **KIND=VIRTUAL,**
IOHFUNCTIONNAME="STREAMIOH"
 - **IOHLIBACCESS=BYFUNCTION** is the default
 - There is no shorthand attribute, like **REDIRECTION**
 - Otherwise, declare like an ordinary record-oriented file
- ◆ **IOHSTRING** attribute
 - Passes parameters to the StreamIOH library
 - Has parameters describing the physical file
 - Has parameters defining the stream/record mapping

Paradigm

MCP-4026 41

To use StreamIOH with a file, you need to declare or file-equate **KIND=VIRTUAL** and **IOHFUNCTIONNAME="STREAMIOH"**. **IOHLIBACCESS=BYFUNCTION** is the default, so this does not need to be specified. Alas, there is no convenient shorthand attribute like **REDIRECTION** as there is for the Redirector.

In all other respects, though, you declare and use the file declaration in your program as you would for an ordinary record-oriented file.

As with the Redirector, the **IOHSTRING** attribute is used to pass parameters to the StreamIOH library. There are two types of parameters used with StreamIOH:

- Parameters describing the physical file
- Parameters defining how the mapping between line-oriented and record-oriented formats is to be done.

Both types of parameters may be intermixed and specified in any order. They are written as a sequence of name=value pairs, separated by commas. Additional spaces may appear around the commas.

IOHSTRING Physical File Parameters

- ◆ General physical file parameters
 - **KIND** = **DISK** | **CDROM** | **VIRTUAL**
 - **EXTMODE** = <mnemonic>
- ◆ Used only with **KIND=VIRTUAL**
 - **IOHTITLE** = <file title>
 - **IOHFUNCTIONNAME** = "<SL name>"
 - **IOHSTRING** = "<string>"
 - **REDIRECTION**

*Note this an **IOHSTRING** parameter inside an **IOHSTRING** attribute*

Paradigm MCP-4026 42

Since the file your program uses is a virtual file, the attributes for that file apply to and are seen by StreamIOH. They do not directly apply to the physical file you are trying to access. StreamIOH passes through many attributes (such as the file name and date/time stamps) between the virtual record-oriented file your program sees and the physical file itself, but in some cases we need to talk about the virtual and physical files individually.

There are several physical file parameters in the **IOHSTRING** attribute that describe characteristics of the physical file to StreamIOH. Note that what we are talking about here are *parameters* within the **IOHSTRING** *attribute*, even though they have the same name as file attributes.

- **KIND** specifies the type of physical file. The choices are **DISK**, **CDROM**, and **VIRTUAL**. The default value is **DISK**.
- **EXTMODE** can be used to override the **EXTMODE** attribute of the physical file. Internally, StreamIOH always processes data as EBCDIC. Your program cannot override this by setting the virtual file's **EXTMODE** attribute.

The previous two parameters can be used with all StreamIOH files. The remaining parameters can be used only when the **KIND** *parameter* has the value **VIRTUAL**, which will be discussed on the next slide.

IOHSTRING KIND Parameter

- ◆ **IOHSTRING KIND=DISK | CDROM**
 - File is local to the MCP environment
 - Located by **FILENAME/TITLE** and **FAMILYNAME**
 - *Do not use "on" in TITLE or LTITLE!*
 - Default is **DISK**

- ◆ **IOHSTRING KIND=VIRTUAL**
 - StreamIOH works through another I/O Handler
 - Must specify **IOHTITLE** or **IOHFUNCTIONNAME**
 - Alternatively, use **REDIRECTION** parameter to specify **REDIRSUPPORT** (the Redirector)
 - Value of the **IOHSTRING parameter** is passed to the other IOH as its **IOHSTRING attribute**

Paradigm

MCP-4026 43

The **KIND** parameter of the StreamIOH **IOHSTRING** attribute determines where and how StreamIOH will be reading or writing the line-oriented text data.

Setting the **KIND=DISK** (the default) or **KIND=CDROM** parameters means that StreamIOH will be accessing a physical file within the MCP file system. That physical file will be located by the file name attribute specified for the virtual file. Note that the family on which the file resides must be specified using the **FILENAME** attribute. Do not use an on-part when specifying the file name using **TITLE** or **LTITLE**, as that implicitly changes the **KIND** of the virtual file to **DISK**, thus making it no longer a virtual file.

Setting the **KIND=VIRTUAL** parameter means that StreamIOH will not be accessing a physical file, at least not directly. Instead it will be accessing the line-oriented data through another virtual file I/O Handler. This could be an IOH that you write, or it could be the Redirector.

- **IOHTITLE** can be used to specify the other IOH by its codefile title.
- **IOHFUNCTIONNAME** can be used to specify the IOH by its SL function name. You should specify the IOH library either by title or by function name, not both.
- **IOHSTRING** can be used to pass parameter information to the other I/O Handler's **IOHSTRING** file attribute. Note that this is an **IOHSTRING parameter** embedded inside an **IOHSTRING attribute**. This can look a little weird when you write it, but as we will see later, it enables some very useful behavior.
- **REDIRECTION** specified as a parameter is equivalent to the **REDIRECTION** file attribute. Specifying this is a shorthand way of specifying the parameters **KIND=VIRTUAL** and **IOHFUNCTIONNAME="REDIRSUPPORT"**.

IOHSTRING Conversion Parameters

- ◆ FILEKIND
- ◆ DATA
- ◆ EXTDELIMITER
- ◆ FORMFEEDISDELIMITER
- ◆ TABINTERVAL
- ◆ MARKID
- ◆ SEQBASE
- ◆ SEQINCREMENT
- ◆ TRIM
- ◆ FOLDING
- ◆ FOLDCHARACTER

Paradigm

MCP-4026 44

The rest of the parameters in a StreamIOH **IOHSTRING** are used to control the conversion between line-oriented and record-oriented formats. These fall logically into a few groups, which I will discuss on the next set of slides.

StreamIOH and FILEKIND

- ◆ **FILEKIND** determines the record length and format of a record-oriented file
 - Text field
 - Sequence number field
 - Mark field
- ◆ StreamIOH determines **FILEKIND** from:
 1. **FILEKIND** parameter in **IOHSTRING** attribute
 2. **FILEKIND** explicitly set for the virtual (record) file
 3. File extension in **FILENAME** or **TITLE** attribute
- ◆ Uses same extensions as Editor/PWB (.C85, .ALG, .WFL, .TXT, etc.)

Paradigm

MCP-4026 45

The **FILEKIND** file attribute specifies the internal layout of records. One of its main uses is to determine the type of compiler associated with a source or object code file. For source files, the **FILEKIND** determines the size and position of the text, sequence number, and mark ID fields in a record.

StreamIOH determines the **FILEKIND** of the virtual file from three places, in the order of descending priority as shown on the slide. If **FILEKIND** has not been specified in some other way, and the file name of the physical file contains an extension (.TXT, .DAT, etc.), then the extension determines the **FILEKIND**. StreamIOH uses the same extension codes as the Editor utility. It also recognizes the extensions used by Programmer's Workbench (PWB, NX/Edit), which all end in "_m".

Basic Record Formatting

- ◆ **FILEKIND=<mnemonic>**
 - Overrides file name extension and **FILEKIND** attribute
 - Determines existence and position of sequence and mark fields
- ◆ **DATA=<integer> [bytes]**
 - Determines length of text area, default = 80
 - **FILEKIND** must resolve to **DATA**, **CDATA**, **CSEQDATA**
 - If **EXTDELIMITER=UNSPECIFIED**
 - **FILEKIND/DATA=*n*** determines the record length
 - On read, fixed-length chunks are passed to the record area without parsing for line delimiters
 - On write, no line delimiter characters are included

Paradigm

MCP-4026 46

StreamIOH determines the **FILEKIND** of the virtual file from multiple places, as discussed on the previous slide. The **FILEKIND** parameter in the **IOHSTRING** overrides any other determination of **FILEKIND** and forces StreamIOH to apply the specified format to the virtual file's record area.

Records in data files do not have sequence and mark ID fields (with the exception of **CSEQDATA**, which has a five-digit sequence number at the beginning of the record) and can be any length up to the system limit of 64K **INTMODE** units. To specify the size of a data record in bytes, the **DATA** parameter is used. If not specified, the default length of data records is 80 bytes. To use the **DATA** parameter, the **FILEKIND** must resolve to one of **DATA**, **CDATA**, or **CSEQDATA**; otherwise an open error will occur.

There is one important twist to accessing data files through StreamIOH. If **EXTDELIMITER** (discussed next) has the value **UNSPECIFIED**, then **FILEKIND** (and for data files, **DATA=*n***) is used to determine the record length. On reads, StreamIOH will read fixed-length chunks from the line-oriented file and pass them to the record area without parsing line delimiters. On writes it will write fixed-length chunks to the line-oriented file without appending line delimiters (which means that the byte stream file will not be line-oriented – it will contain fixed-length records written head-to-tail).

Delimiter Character Handling

- ◆ **EXTDELIMITER=**
 - **UNSPECIFIED** | **NL** | **CR** | **LF** | **CRLF** | **CRCC**
 - May need to set this for data files
 - Default **UNSPECIFIED** implies:
 - No line delimiters recognized
 - Reads fixed-length "records" from the stream file
- ◆ **FORMFEEDISDELIMITER=TRUE | FALSE**
- ◆ **TABINTERVAL=<integer>**
 - Expands tab (HT) characters to spaces
 - **TABINTERVAL=0** disables tab expansion
 - Operates only when reading from stream file
 - Does not replace spaces with tabs on write

Paradigm MCP-4026 47

The StreamIOH **EXTDELIMITER** parameter echoes the file attribute of the same name. **EXTDELIMITER** specifies the type of line delimiter used in a line-oriented text file. The mnemonic values are

- **UNSPECIFIED**
- **NL** (newline)
- **CR** (carriage-return)
- **LF** (line-feed)
- **CRLF** (carriage-return followed by line-feed, the Windows/DOS standard)
- **CRCC** (carriage-return followed by either line-feed or form-feed)

This is a physical file attribute and is stored in the MCP disk directory. StreamIOH will assume the value read from the disk directory unless it is overridden by a logical file attribute setting or the **EXTDELIMITER** parameter in **IOHSTRING**.

Once again, if **EXTDELIMITER** has the value **UNSPECIFIED**, StreamIOH will not know how to parse for line delimiters, so the physical file will be assumed to not contain delimiters, and StreamIOH will simply read and write fixed length chunks of data having a length determined by the size of the virtual file record.

The **FORMFEEDISDELIMITER** parameter specifies whether form-feed (ASCII hex 0C) characters in the text stream will be recognized as line delimiters. The default is **FALSE**.

The **TABINTERVAL** parameter specifies whether horizontal tab (ASCII hex 09) characters in the text stream will be expanded to spaces. If this parameter is not specified, or has the value zero, tab characters are passed through as part of the text. If the parameter has a non-zero value, tab characters are replaced with spaces to pad out the record to the next multiple of the interval. A very popular tab interval is eight. The default value is zero.

Note that **TABINTERVAL** is only effective when reading from the line-oriented file. It does not cause "tab compression" when writing to a line-oriented text file. All (non-trailing) spaces are passed through on a write without being converted to tabs.

Sequence and Mark Filling

- ◆ Only apply if reading from the stream file
- ◆ New mark ID and sequence value overwrite any converted from the stream file
- ◆ Size and position determined by the effective **FILEKIND** value
- ◆ **MARKID**=" <string> "
- ◆ **SEQBASE**=<integer>
- ◆ **SEQINCREMENT**=<integer>

Paradigm

MCP-4026 48

For effective **FILEKIND** values that represent record formats having sequence number and mark ID fields, StreamIOH can supply values for these fields when records are read. Any sequence number or mark ID in the line-oriented file will by default be passed through to the record area unless stripped off by the **TRIM** parameter, discussed next.

If **SEQINCREMENT** is specified, the sequence numbers generated by StreamIOH will override those from the line-oriented file. Similarly, if **MARKID** is specified, that value overrides any from the line-oriented file. **SEQBASE** is only valid if **SEQINCREMENT** has a non-zero value.

The default **MARKID** is a null string. The default for both **SEQBASE** and **SEQINCREMENT** is 100.

Record Trimming

- ◆ Trims sequence field, mark field, and/or trailing blanks when writing to stream file
- ◆ Drops sequence or mark field from record on read
- ◆ **TRIM=**
 - **NONE**
 - **BLANKS**
 - **SEQUENCE**
 - **ID**
 - **ALL** (default)

Paradigm MCP-4026 49

StreamIOH has the ability to trim portions of a record as it is transferred between the line-oriented file and the record area. Trimming is most often useful when writing to a line-oriented file. For **FILEKIND** values that support sequence numbers, you can choose to trim them from the record before the text is transferred to the line-oriented file. For all types of files, you can choose to trim trailing blanks from the line-oriented file. The default is to trim everything.

Trimming can also be applied when reading from a line-oriented file. For **FILEKIND** values that support sequence numbers, this causes the sequence and mark ID from the line-oriented file to be dropped and values generated by StreamIOH to be placed in the record area.

The **TRIM** parameter has the following options:

- **NONE** – trim nothing.
- **BLANKS** – trim trailing blanks from the text area of the record, along with any sequence or mark ID field present to the right of the text area.
- **SEQUENCE** – trim the sequence number from the record area. There are two alternatives when this is specified:
 - If the sequence number is to the left of the text area (as in COBOL), only the sequence number field is trimmed.
 - If the sequence number is to the right of the text area, the sequence and mark ID (if present) will be trimmed.
- **ID** – trim the mark ID field, if any.
- **ALL** – trim everything – sequence number, mark ID, and trailing blanks.

Record Folding

- ◆ Controls mapping of long lines in the stream file to the record file
 - On read, long lines from the stream file may be *folded* into multiple records
 - On write, multiple records may be *unfolded* into a single long line for the stream file
 - "Fold character" is placed at the end of folded records
 - Default is **FOLDING=BLIND**
- ◆ Trimming and folding can conflict
 - It's complex – see documentation for specifics
 - Trimming is always applied before folding

Paradigm

MCP-4026 50

With StreamIOH, the record area must be defined for the virtual file with a fixed maximum length. On the line-oriented side, however, the lines can be of arbitrary length. For some files, accommodating very long lines may make the record area inconveniently large.

To deal with this, StreamIOH supports folding of long lines into multiple records. As long lines are read from the line-oriented file, they can be broken into pieces no longer than the size of the record area. A "fold character" may optionally be placed at the end of the folded text in the record area to indicate that the record has been folded and that more data from the same line will appear in subsequent records. If a line is short enough not to require folding (or it is the last piece of a folded line), no fold character is placed in the record area.

The documentation indicates that the default type of folding is **TRUNCATE**. That is incorrect – the default is actually **BLIND**.

When writing to a line-oriented file, StreamIOH also supports unfolding multiple records to create one long line. The fold character (if specified) is used to determine where in the record area the fold occurs. The process of folding and unfolding records is symmetric, so that a record-oriented file created by folding may be unfolded to recreate the original line-oriented file.

Some choices for the **FOLDING** and **TRIM** parameters can conflict. The interaction between these two parameters is too complex to delve into here. See the documentation in the *I/O Subsystem Guide* on the **FOLDING** option for more information. Keep in mind, though, that trimming is always applied by StreamIOH before it considers a record for folding.

Folding Parameters

- ◆ **FOLDING=NONE | SEQUENCE | ID |**
 BLIND | SPACE | TRUNCATE

- ◆ **FOLDCHARACTER=**
 - **NONE** (default)
 - **AMPERSAND (&)**
 - **ATSIGN (@)**
 - **BACKSLASH (\)**
 - **DOLLARSIGN (\$)**
 - **NUMBERSIGN (#)**
 - **PERCENTSIGN (%)**
 - **SLASH (/)**
 - **VERTICALLINE (|)**

Paradigm MCP-4026 51

The **FOLDING** parameter has the following options:

- **NONE** – no folding or unfolding is done. On read, if a line is encountered in the line-oriented file that is longer (after trimming is applied) than the file's record area, an I/O **DATAERROR** exception is returned to the MCP application program.
- **SEQUENCE** – the lines in the line-oriented file are assumed to be of the exact fixed size defined by the effective **FILEKIND** to contain the text area and sequence number fields. A **DATAERROR** exception occurs if a line is encountered that that conflicts with this assumption.
- **ID** – the lines in the line-oriented file are assumed to be of the exact fixed size defined by the effective **FILEKIND** to contain the text area, sequence number, and mark ID fields. A **DATAERROR** exception occurs if a line is encountered that that conflicts with this assumption.
- **BLIND** – lines in the line-oriented file are folded in increments that are the size of the record area. If a fold character is specified, it is inserted in the last character position of the record area. This is the default setting (the documentation through MCP 11.1 states it is **TRUNCATE** – that is incorrect).
- **SPACE** – lines in the line-oriented file are folded in increments no larger than than the size of the record area. The fold occurs at a point where there is a transition between space and non-space characters. The fold character (if specified) is inserted in the text area in place of the last blank character, and the rest of the text area field is blank.
- **TRUNCATE** – lines are not folded at all. If a line is read from the line-oriented file that is longer than the record area, it is simply truncated on the right to fit in the record area.

The **FOLDCHARACTER** parameter determines the character to be inserted into the record area at the point that the fold occurred. It can be any of the choices shown on the slide. The default is **NONE** (the documentation is wrong about this, too).

Special Considerations

- ◆ For purely sequential I/O, stream/record conversion is done on the fly
- ◆ If a random I/O request occurs
 - Entire stream file is read and converted to a temporary (hidden) fixed-length record file
 - All further I/O is against this temporary file
 - When the virtual file is closed, stream file may be rewritten from temporary file to apply any updates
- ◆ StreamIOH works internally in EBCDIC
 - Translation may occur between physical file and IOH
 - May also occur between IOH and virtual (record) file

Paradigm

MCP-4026 52

There are a couple of special considerations when using StreamIOH.

First, we have been talking thus far about using StreamIOH for sequential access to a line-oriented byte stream file. When used in this fashion, all conversion between the line-oriented and record-oriented format is done on the fly as reads and writes are issued by the MCP application.

StreamIOH also supports random I/O against the line-oriented file. The problem with random I/O is that there is no way to predict where lines in the line-oriented file occur, since they can be variable length. To handle this, the first random I/O operation causes StreamIOH to read the entire stream file, converting it into a temporary, hidden, fixed-length record file. All further I/O for the virtual file will occur against this temporary file. When the virtual file is closed, the temporary file is discarded. If writes occurred to the temporary file, it is used to recreate the line-oriented file before being discarded.

The second consideration is that StreamIOH always works internally using the EBCDIC character code. Depending on the **INTMODE** of the virtual file, translation may occur between the MCP application's record area and StreamIOH. Depending on the **EXTMODE** of the physical file, translation may also occur between StreamIOH and the physical file. In most cases involving line-oriented text files, however, this translation something you want to have happening.

More StreamIOH Examples

```
RUN MY/PROG; FILE FIXED (  
  KIND=VIRTUAL, IOHFUNCTIONNAME="STREAMIOH",  
  LFILENAME=MSTREAM.TXT, FAMILYNAME=PACK,  
  IOHSTRING="FILEKIND=DATA, DATA=256," &  
    "FOLDING=TRUNCATE" );  
  
RUN YOUR/PROG; FILE SOURCE (  
  KIND=VIRTUAL, IOHFUNCTIONNAME="STREAMIOH",  
  FILENAME=UPDATE.C85, FAMILYNAME=DEV,  
  IOHSTRING="KIND=CDROM," &  
    "TRIM=ID, FOLDING=NONE," &  
    "SEQBASE=100100, SEQINCREMENT=100," &  
    "MARKID=" "UPDATE" " " );
```

Paradigm

MCP-4026 53

This slide shows two more examples of file attribute equations that invoke StreamIOH. In the first example, the line-oriented file is considered to be just textual data, and is converted to a fixed-length, 256 character record area.

In the second example, a line-oriented file is read from CDROM and converted to COBOL-85 record format (as determined by the extension on the file name). Any mark ID from the line-oriented file is discarded and replaced by the literal **"UPDATE"**. The records are resequenced 100100+100 as they are read. Since **FOLDING=NONE**, if lines longer than the COBOL-85 record length are read, a **DATAERROR** is returned to the MCP application. The **TRIM=ID** parameter is redundant in this example, as the **MARKID** parameter causes any mark ID field read from the line-oriented file to be replaced according to the parameter value supplied.

[Demo here]

Using StreamIOH with Redirector

- ◆ StreamIOH can use Redirector as its "physical file"
- ◆ Allows MCP apps to access text files on remote shares as record-oriented files
- ◆ In StreamIOH's **IOHSTRING**:
 - Specify the **REDIRECTION** parameter
 - Specify the the Redirector **IOHSTRING *attribute*** parameters in the StreamIOH **IOHSTRING *parameter***
 - Specify StreamIOH's **FILENAME/TITLE** attribute as the absolute or relative UNC path to the remote file
 - Can use credentials and NXCONFIG files

Paradigm

MCP-4026 54

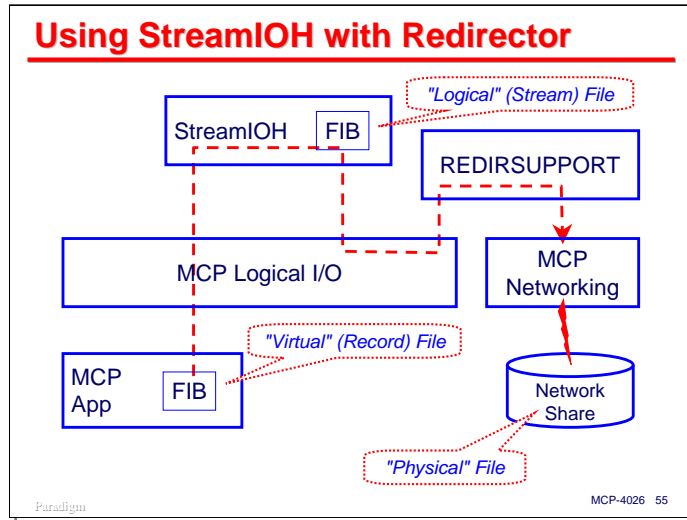
Since StreamIOH can be configured to use another virtual file I/O Handler to access the line-oriented data, it is possible to team StreamIOH with the Redirector. This allows you to easily configure MCP applications so they access line-oriented text files directly on network shares. StreamIOH provides some special **IOHSTRING** parameters to make this convenient.

You can invoke the Redirector through StreamIOH by specifying **KIND=VIRTUAL** and **IOHFUNCTIONNAME="REDIRSUPPORT"** as parameters in the StreamIOH **IOHSTRING** attribute. Alternatively, you can just specify the **REDIRECTION** parameter in **IOHSTRING**, which will set the other two parameters for use with the Redirector.

The problem with using both StreamIOH and Redirector together is that they both generally require parameters to be specified in their **IOHSTRING** attributes. StreamIOH addresses this by providing an **IOHSTRING *parameter*** inside its **IOHSTRING *file attribute***. The string specified with the **IOHSTRING *parameter*** is simply passed by StreamIOH to the Redirector's **IOHSTRING *file attribute***. The main thing you need to be careful of with this feature is that you count all the quotes correctly when you specify the attribute values.

The file name attribute (**FILENAME**, **LTITLE**, etc.) is simply passed by StreamIOH to the Redirector. You can use either the relative path naming method, or the ***UNC** absolute path naming method.

You can also use credentials file and **NXCONFIG** files with the StreamIOH/Redirector combination the same way you do with Redirector alone.



This slide shows how StreamIOH and the Redirector are combined to allow an MCP application to read or write line-oriented byte stream files on a network share as if they were record-oriented files.

The MCP application opens a virtual file that connects to the StreamIOH library. From the **KIND=VIRTUAL** and **IOHFUNCTIONNAME** parameters in its **IOHSTRING** attribute, StreamIOH knows to configure its internal file declaration as a Redirector file.

The **IOHSTRING** parameter supplied by the application in the virtual file's **IOHSTRING** attribute is used to set the value of the Redirector's **IOHSTRING** attribute. This gives Redirector the information necessary to open an SMB connection to the network share and access the remote line-oriented file.

StreamIOH/Redirector Example

```
RUN MY/PROG; FILE FIXED (  
  KIND=VIRTUAL, IOHFUNCTIONNAME="STREAMIOH",  
  LFILENAME=SUBDIR/MYSTREAM.TXT,  
  IOHSTRING="REDIRECTION," &  
    "FILEKIND=DATA, DATA=120," &  
    "EXTDELIMITER=NL," &  
    "IOHSTRING=" "SERVER=WINFS1 " &  
      "SHARE=PUBLIC TIMEOUT=15 " &  
      "CREDENTIALS=FOO/BAR" " " );
```

StreamIOH IOHSTRING attribute

Redirector IOHSTRING attribute

Paradigm MCP-4026 56

This slide shows an example of how you would declare the virtual file when teaming StreamIOH with Redirector. Everything is the same as we have seen previously for StreamIOH file attributes, except for two things in the **IOHSTRING** attribute

- The **REDIRECTION** parameter is specified to cause StreamIOH to read and write from a Redirector file instead of a physical file within the MCP environment.
- The **IOHSTRING** *parameter* is specified inside the **IOHSTRING** *attribute*. The value of this parameter is passed to the Redirector, which uses it to establish the connection to the network share.

[Demo here]

References

- ◆ MCP I/O Subsystem Guide
 - Section 28 – Virtual Files
 - Section 29 – The Redirector
 - Section 30 – StreamIOH
- ◆ MCP File Attributes Reference Manual
- ◆ "Using Stream Files" UNITE presentation
 - <http://www.digm.com/UNITE/2001>
- ◆ "Using Stream Files" article in Gregory's *ClearPath/A-Series Technical Journal*, volume 16 #1, Jan/Feb 2002

Paradigm

MCP-4026 57

Finally, here are several references to more information on the Redirector and StreamIOH.

- The primary resource for these facilities is the *MCP I/O Subsystem Guide*, sections 28 through 30.
- Since byte stream files, the Redirector, and StreamIOH are all configured using file attributes, the *File Attributes Reference Manual* is another valuable resource.
- I presented a detailed talk on stream files at the 2001 UNITE conference in Phoenix. That presentation and several sample programs are available on our web site at the link shown.
- Don Gregory later invited me to turn that presentation into an article, which was published in the January/February 2002 issue of his *Technical Journal*.

More References

- ◆ "The Care and Feeding of IOHANDLERS"
(2 articles) in Gregory's *ClearPath/A-Series Technical Journal*, volume 16 #7, Oct/Nov 2002
- ◆ Sample files for this presentation
 - <http://www.digm.com/UNITE/2007>

Paradigm

MCP-4026 58

Here are some additional references:

- Don Gregory wrote a pair of articles on using and programming for virtual file I/O Handlers, which were published in the October-November 2002 issue of his *Technical Journal*.
- This presentation, along with several sample programs and WFL jobs that illustrate use of both the Redirector and StreamIOH will be available on our web site shortly after the conference ends at the link shown on the slide.

END

Using Redirector and StreamIOH

2007 UNITE Conference
Session MCP-4026